$1.80

# Apple Assembly Line

Volume 5 -- Issue 9 June, 1985



## S-C Macro Assembler ProDOS

We will begin shipping the ProDOS version of the S-C Macro Assembler in July, so we are now accepting advance orders. There is more to the ProDOS version than just a change of operating systems. The new upgrade includes a couple of major new features:

.INB (INclude Blocked) directive -- This works just like .IN, except that only one disk block at a time is overlaid into memory. Allows assembly of much larger files, with only a minor speed penalty.

.AC (Ascii Compressed) directive -- Generates compressed strings from a string between delimiters, according to rule tables. Very complex, but worth the effort if you have a lot of messages and need to save memory.

The price of the ProDOS version alone will be $100. The upgrade from DOS Version 2.0 to ProDOS will be $30. The upgrade from DOS Version 1.0 or 1.1 to ProDOS will be $50, and will include DOS Version 2.0. The initial purchase price of the DOS 3.3 and ProDOS versions together will be $120. These are introductory prices which may well be raised in a few months.

## 65802's Are Here!

After many months of manufacturing delays, Western Design Center is shipping 65802 and 65816 microprocessors. We recently received a final production '802, and it's now happily processing away in Bob's oldest Apple II (#219). You can order the chips from WDC for $95.00. Call (602) 962-4545.

The Boyer-Morris String Search Algorithm..........Bob Bernard
Westport, CT

For years now, I have been working on a debugger for the Apple. Lately I have been adding a hex string search capability to it.

I needed one so I could look through the Apple IIc (ProDOS) utilites to see how it squirrels away in the alternate page screen holes user specified default settings for the serial ports. These are used at PR#1 or 2 time to simulate the dip switches on the Super Serial Card in a IIe. Without setting them you always get 9600 bps, etc. (Imagewriter settings, that is). I (and I assume other AAL readers) want a little routine for DOS 3.3 hello that will allow the user's defaults to be put away the same as the IIc utility does.

Well, that routine is not ready yet. However, the search utility is rather interesting in its own right.

I was just going to code up a straight hex search, but then I mentioned it to my computer science graduate son, David. He was horrified that I would waste my time on anything so crude. That's what I get for bringing up a programmer! David insisted that I should instead code an implementation of Boyer and Moore's algorithm, which appeared in the October 1977 issue of the Communications of the ACM. [A more recent reference is in the book "Algorithms", by Robert Sedgewick, (Addison-Wesley Publishing Co., 1983, 551 pages) on pages 249-252.]

Well, I read the article and it seemed like a challenge. Besides it looked like a real time saver, and could also be used for character string searches. The code here has been excerpted from my debugger, and then worked over by Bob S-C.

The "conventional" or "brute-force" search technique aligns the search pattern with the left end of the string to be searched through and compares one byte at a time, from left to right, until either the entire pattern is compared successfully or a mismatch occurs. In the latter case the search window is moved one byte to the right, and the comparing process is repeated.

Without any knowledge about the contents of the search pattern, the most the window can be moved is one place to the right. Boyer-Moore owes its speed advantage to the fact that it uses context (i.e. knowledge about the contents of the pattern to be searched for) to increase the distance that the search window can be advanced when a mismatch occurs. Thus efficiency increases as the length of the pattern increases, which does not happen in a conventional search.

The cost of this benefit (there always is a cost) is that a table (called DELTA1 in the CACM article and DELTA.TABLE in my program) is required to store this context information, 256 bytes in this implementation. One byte is needed in the table for every possible value of the characters in the string to be searched.

If a particular byte appears in the search pattern, then the

```
S-C Macro Assembler Version 2.0.....................................$100
Version 2.0 Upgrade Kit for 1.0/1 1/1 2 owners.......................$20
Source Code for Version 1 1 (on two disk sides).....................$100
Full Screen Editor for S-C Macro (with complete source code)..........$49
S-C Cross Reference Utility (without source code).....................$20
S-C Cross Reference Utility (with complete source code)...............$50
DISASM Dis-Assembler (RAK-Ware).......................................$30
Source Code for DISASM...................................additional $30
S-C Word Processor (with complete source code)........................$50
DP18 Source and Object................................................$50
Double Precision Floating Point for Applesoft (with source code)......$50
S-C Documentor (complete commented source code of Applesoft ROMs).....$50
Source Code of //e CX & F8 ROMs on disk...............................$15

(All source code is formatted for S-C Macro Assembler. Other assemblers
require some effort to convert file type and edit directives.)

AAL Quarterly Disks...........................each $15, or any four for $45
                                   Jan-Mar    Apr-Jun    Jul-Sep   Oct-Dec
 Each disk contains        1980       -          -          -         1
 the source code from      1981       2          3          4         5
 three issues of AAL,      1982       6          7          8         9
 saving you lots of        1983      10         11         12        13
 typing and testing.       1984      14         15         16        17
                           1985      18         19

AWIIe Toolkit (Don Lancaster, Synergetics)............................$39
ES-CAPE: Extended S-C Applesoft Program Editor (new price, was $60)  $40
"Bag of Tricks", Worth & Lechner, with diskette.............($39.95)  $36
MacASM -- Macro Assembler for MacIntosh (Mainstay).........($150.00) $100

Blank Diskettes (Verbatim)......................... package of 20 for $32
    (Premium quality, single-sided, double density, with hub rings)
Vinyl disk pages, 6"x8.5", hold two disks each..................10 for $6
Diskette Mailing Protectors (hold 1 or 2 disks).............40 cents each
   (Cardboard folders designed to fit 6"X9" Envelopes.)    or $25 per 100
Envelopes for Diskette Mailers...........................   6 cents each

quikLoader EPROM System (SCRG)................................($179) $170
PROmGRAMER (SCRG)...........................................($149.50) $140
D MAnual Controller (SCRG).....................................($90)  $85
Switch-a-Slot (SCRG)..........................................($190) $175
Extend-a-Slot (SCRG)...........................................($35)  $32


"Apple ProDOS: Advanced Features for programmers", Little..($17 95)   $17
"Inside the Apple //c", Little.............................($19.95)   $18
"Inside the Apple //e", Little.............................($19.95)   $18
"Apple II+/IIe Troubleshooting & Repair Guide", Brenner....($19.95)   $18
"Apple ][ Circuit Description", Gayler.....................($22.95)   $21
"Understanding the Apple II", Sather.......................($22.95)   $21
"Understanding the Apple //e", Sather......................($24.95)   $23
"Enhancing Your Apple II, vol. 1", Lancaster...............($15.95)   $15
"Enhancing Your Apple II, vol. 2", Lancaster...............($17 95)   $17
"Assembly Cookbook for the Apple II/IIe", Lancaster........($21.95)   $20
"Beneath Apple DOS", Worth & Lechner.......................($19.95)   $18
"Beneath Apple ProDOS", Worth & Lechner....................($19.95)   $18
"6502 Assembly Language Programming", Leventhal............($18.95)   $18
"6502 Subroutines", Leventhal..............................($18.95)   $18
"Real Time Programming -- Neglected Topics", Foster.........($9.95)    $9
"Microcomputer Graphics", Myers............................($12.95)   $12
"Assem  Language for Applesoft Programmers", Finley & Myers.($16.95)  $16
"Assembly Lines -- the Book", Wagner ......................($19.95)   $18
"AppleVisions", Bishop & Grossberger.......................($39.95)   $36

Add $1 50 per book for US shipping.  Foreign orders add postage needed.

          Texas residents please add 6 1/8 % sales tax to all orders.


          *** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
          ***                  (214) 324-2050                  ***
          *** We accept Master Card, VISA and American Express ***
```

corresponding DELTA table entry contains the distance that the rightmost occurrence of that byte is from the left end of the pattern. All other entries contain the value -1. When a mismatch occurs, the DELTA table entry corresponding to that byte from the text being searched is used to compute how far to advance the search window. If that byte does not appear anywhere in the pattern, then the search window can be advanced by the full length of the pattern.

Since moving the search window, and the associated testing for finished, take most of the time in any searching technique, saving time here can be extremely beneficial, and explains why Boyer and Moore should be complimented.

My program uses the control-Y monitor command, in the form

```
     adr1.adr2^Y <hexstring>
```

The two addresses specify the start and end of the area to be searched. "^Y" stands for "control-Y". The hex string may be separated from the control-Y by one or more spaces, if you desire. Since the control-Y doesn't show up on the screen, I usually type at least one space before the hex string. The hex string itself is a continuous string of hex digits, with no imbedded spaces. Here is an example that will search from $800 to $BFFF for "BERNARD":

```
     800.BFFF^Y 4245524E415244
```

The program will list the starting addresses of any and all complete matches that are found.

The maximum length of the hex string is limited by the monitor input buffer. Since the longest command you can type is less than 256, and you have to use around ten characters for the addresses and control-Y, that puts an upper limit of less than 246 hex digits in your command. Each byte of the search pattern (or "key") is made up of two hex digits, so the maximum hex string will be less than 123 bytes long.

I assigned DELTA.TABLE to the area $02D0.03CF. Since I scan and collect the search pattern right in the monitor keyboard buffer at $0200, after converting to hex bytes it will run no higher than $027F.

Actually, I only implemented a simplified version of Boyer and Moore's procedure. The CACM article also discusses a second table, DELTA2, which is filled with additional context information regarding "terminating substrings" of the search pattern. In cases where a partial mismatch occurs, it may be possible to advance the search window farther than the DELTA1 table would indicate. However, since such situations occur in less than 20% of the cases, David allowed that the potential additional speed did not justify the time and effort and the additional table and code space that would have been required, and he gave me a passing grade on my effort without it. The incorporation of this additional capability, and changes to make the program an ASCII search, are left "as a exercise for

the reader."

My program must go through several steps. First it has to find and pack up the search key. Next it must build the DELTA table. And finally the search can be performed.

Lines 1290-1360 will be executed when you BRUN the program. They install the control-Y vector and jump into the monitor, just as though you entered with CALL-151.

When you enter the search command, the Apple monitor parses the command line up to and including the control-Y, and then branches to my code at line 1380. The two addresses will have been converted and stuffed into A1 ($3C,3D) and A2 ($3E,3F). A variable named YSAV (at $34) contains the index to the next character following the control-Y.

Lines 1400-1440 skip over any blanks you may have typed between the control-Y and the first hex digit. Actually, the Y-register gets incremented once too often, so lines 1460-1470 decrement Y and save it; now YSAV points to the first hex digit in the search key.

The next problem I had to solve was to differentiate odd from even length strings and arrange them properly, adding a leading zero when an odd number of hex digits is input. Lines 1490-1530 search for the end of the hex string; if there are no digits at all, we are finished and line 1530 returns for the next monitor command.

This is a nice place to insert a brief description of the NXTCHAR subroutine, found in lines 2460-2590. NXTCHAR picks up the next character from the input buffer, and tests to see if it is a hex digit. If so, it returns either $00-09 or $FA-FF in the A-register, and carry will be clear. If not a hex digit, it returns with carry set. If we got a digit, the Y-register indexing the input buffer will have been advanced.

Lines 1550-1590 compute the key length. Since two digits make a byte, the number of digits in the hex string divided by two gives the number of bytes. But I actually want to use the byte-count-minus-one. Also I need to adjust for odd or even length strings. Lines 1600-1650 take care of these details. If the count was odd, I jump into the middle of the packing loop so that a leading zero gets inserted.

Lines 1670-1800 comprise the packing loop. NXTCHAR will return with carry set when we try to get a digit beyond the end of the key, so line 1680 is the only test in the loop. Lines 1670-1730 retrieve a left-hand digit and store it in the buffer. Lines 1740-1800 do the same for right-hand digits. Key bytes are stored starting at $0200, so they never catch up to the advancing retrieval of digits.

Line 1810 sets YSAV to point to the first character past the end of the hex string. This will usually be a carriage return, or another monitor command. Unless it is beyond $2CF, the monitor will correctly continue parsing whatever is in the

buffer when we are through searching. At $2D0 and beyond, the DELTA table will clobber any further characters.

Now we come to the Boyer-Moore part. Lines 1820-1870 initialize the DELTA table to all -1 values, which is what we want for any bytes not present in the key. When the loop finishes, X=0 again.

Lines 1880-1970 scan through the search key from left to right, and store into DELTA the index of the rightmost occurrence of each value in the key. For example, if the key is "4245524E415244" ("BERNARD" again), the DELTA values will be:

```
     DELTA+$41:  4
     DELTA+$42:  0
     DELTA+$44:  6
     DELTA+$45:  1
     DELTA+$4E:  3
     DELTA+$52:  5 (also at 2, but 5 is rightmost)
    all others:  -1
```

We'll continue with this example after a brief look at the rest of the code.

Lines 1980-2040 back up the end pointer, which has been patiently waiting all this time in A2L and A2H. We subtract the key length (in bytes, not digits) from the end pointer, so that we will not try to match the key any further than necessary. We could do this inside the search loop, but it will run faster if we do it once before the loop.

Lines 2050-2440 perform the search. I inserted lines 2070-2110 inside the loop to printout the search window start address each time through the loop. This helps me to make sure it is working, and to explain how. Of course you should remove these five lines before using the routine for real problems. Notice they are all marked "<<<DEBUG>>>".

Lines 2120-2170 check whether the beginning of the search window has moved past the end of the area to be searched. If so, we are finished.

Lines 2180-2240 compare bytes from the key and the search window. If the entire key matches, we fall out of the loop into lines 2250-2300, where the address of the match will be printed. After a successful match the search window will be moved one byte to the right by lines 2370-2430, and we will begin the SEARCH.LOOP again.

Notice that the key is compared from right-to-left, not left-to-right. This is a critical part of the Boyer-Moore method. If a key byte does not match a search-window byte, we branch to line 2320. The byte from the search window is in the A-register. Lines 2320-2370 compute how far we can advance the search window, based on just what character we DID find in the search window, and how far into the key we had already matched.

To see how this works, let's continue the "BERNARD" example.

Suppose the text we are searching is "THERE ARE FEW ST. BERNARDS IN SAN BERNARDINO." The key will be BERNARD, entered in hex as shown above. We first try to match BERNARD at the beginning of the text. We start at the right end, matching the "D" of the key with "A" of the text. The match fails, so we look up the "A" value in the DELTA table, which is 4. We subtract the delta value (4) from the current key index (6) and add the result (6-4=2) to the search window address. Note that this has the result of aligning the "A" of BERNARD with the "A" in the text.

Back to the top, and we now try to match the "D" of BERNARD to the "E" at the end of "ARE". Failure again! This time the DELTA value is 1, and we are still at position 6 in the key: index-delta is 5, so we advance the window by 5. This lines up the "E" of BERNARD with the E of the text. The next attempted match will find a blank in the text, which does not occur in the key at all. The delta value for blank is -1: 6-(-1)=7, so we will advance the window by 7. Now the window is up to "ST. BER" in the text.

When we compare "D" of BERNARD to "R" in the text, we fail again. The delta value for R is 5. There are two R's in BERNARD, but the rightmost one is at index 5. We can move the search window by 6-5=1. Next we try "D" against "N". The delta value of "N" is 3, so we can move the window 6-3=3 bytes. This time we have found "BERNARD"!

If you count it all up, we have compared the "D" of BERNARD with only six characters, and already we are at the first occurrence of the whole key in the text. A conventional search would have tried to match the first character of the key ("B") with all 18 characters in the text which precede the first "B" of the text. We have saved 13 times around the main loop! Of course, our loop is a tiny bit longer, but the end result is faster.

Here is a step-by-step picture of the entire search, which finds BERNARD twice:

```
THERE ARE FEW ST. BERNARDS IN SAN BERNARDINO.
BERNARD
  BERNARD
       BERNARD
              BERNARD
               BERNARD
                  BERNARD  (success!)
                   BERNARD
                          BERNARD
                             BERNARD
                                  BERNARD  (success!)
                                   BERNARD
                                          BER... (end)
```

I have tacked two more examples onto the end of the source code, at lines 2620-2690. You can play with them. The five <<<DEBUG>>> lines will print out the window address at each step, so you can see how the search progresses. Remember to

take those lines out before you make a production version of the program.

If you decide to include this search algorithm in your own private debugger program, like I am, you might want to add the ability to use an ASCII string for the key. You could use a quotation mark after the control-Y to signal the packer loop that an ASCII string follows. You might also want to add single-byte wildcard characters, and/or the ability to ignore the high-order bit of each byte matched.

Perhaps the Boyer-Moore algorithm would be even more useful in a data base program, a word processor, or other context in which you are searching through huge quantities of text for relatively interesting keys. My example should get you started, and my son will be proud of you!

```
                  *SAVE S.HEX.SEARCH
                  *--------------------------------
                  *     MEMORY SEARCH FOR HEX STRING
                  *         BY BOB BERNARD, MAY 17, 1985
                  *         MODIFIED BY BOB S-C, MAY 27TH
                  *     ADR1.ADR2^YXXXXXXXXXXX
                  *         ("^Y" MEANS CONTROL-Y)
                  *
                  *     SEARCH MEMORY FROM ADR1 THRU ADR2
                  *     LOOKING FOR REFERENCES TO
                  *     THE HEX STRING, XXXXXXXXXX
                  *
                  *--------------------------------
34-               YSAV        .EQ $34
3C-               A1L         .EQ $3C,3D     START OF SEARCH AREA
3E-               A2L         .EQ $3E.3F     END OF SEARCH AREA
40-               KEY.LENGTH  .EQ $40        (MONITOR'S A3L)
                  *--------------------------------
0200-             KBDBUF      .EQ $0200 THRU $2CF
02D0-             DELTA.TABLE .EQ $02D0 THRU $3CF
03F8-             USRADR      .EQ $03F8   CTL-Y JUMPS HERE
                  *--------------------------------
F941-             PRINTAX     .EQ $F941
FD8E-             CROUT       .EQ $FD8E   NEW LINE
FF69-             MONZ        .EQ $FF69   MONITOR, NO BELL
                  *--------------------------------
                         .OR $0800
                         .TF B.HEX.SEARCH
                  *--------------------------------
                  HEX.SEARCH
0800- A9 4C                LDA #$4C       JMP OPCODE
0802- 8D F8 03             STA USRADR       STUFF INTO CNTL-Y EXIT LOC
0805- A9 12                LDA #SEARCH      LO ADR
0807- 8D F9 03             STA USRADR+1
080A- A9 08                LDA /SEARCH      HI ADR
080C- 8D FA 03             STA USRADR+2
080F- 4C 69 FF             JMP MONZ         MONITOR, NO BELL
                  *--------------------------------
                  SEARCH
                  *---SKIP LEADING BLANKS----------
0812- A4 34                LDY YSAV         NEXT VALID KBDBUF CHAR
0814- B9 00 02    .1       LDA KBDBUF,Y     GET CHAR FROM
0817- C8                   INY              KEYBOARD BUFFER
0818- C9 A0                CMP #" "         SKIP LEADING BLANKS
081A- F0 F8                BEQ .1
                  *---MARK KEY START---------------
081C- 88                   DEY
081D- 84 34                STY YSAV         WHERE SCAN STARTS
                  *---FIND END OF KEY--------------
081F- 20 C2 08    .2       JSR NXTCHAR
0822- 90 FB                BCC .2           ...HEX DIGIT
0824- C4 34                CPY YSAV         CHECK FOR NULL KEY
0826- D0 01                BNE .3           ...NOT NULL
0828- 60                   RTS              NULL KEY
```

```
                  *---COMPUTE KEY LENGTH----------
0829- 98          .3        TYA
082A- E5 34                 SBC YSAV

082C- 4A                    LSR
082D- 85 40                 STA KEY.LENGTH
082F- A4 34                 LDY YSAV
0831- A2 00                 LDX #0
0833- 8E 00 02              STX KBDBUF       (IN CASE ODD COUNT)
0836- B0 0E                 BCS .5           ...ODD NUMBER OF BYTES
                  *---ADJUST FOR EVEN LENGTH------
0838- C6 40                 DEC KEY.LENGTH MAKE EVEN LENGTH ONE LESS
                  *---LEFT NYBBLE----------------
083A- 20 C2 08    .4        JSR NXTCHAR
083D- B0 15                 BCS .6           END OF KEY
083F- 0A                    ASL
0840- 0A                    ASL
0841- 0A                    ASL
0842- 0A                    ASL
0843- 9D 00 02              STA KBDBUF,X     LEFT HALF DEST CHAR
                  *---RIGHT NYBBLE---------------
0846- 20 C2 08    .5        JSR NXTCHAR
0849- 29 0F                 AND #$0F
084B- 1D 00 02              ORA KBDBUF,X     MERGE HI NIBBLE
084E- 9D 00 02              STA KBDBUF,X
0851- E8                    INX
0852- D0 E6                 BNE .4           ...ALWAYS
0854- 84 34       .6        STY YSAV
                  *---INIT ALL DELTAS=-1 ---------
0856- A2 00                 LDX #0
0858- A9 FF                 LDA #-1
085A- 9D D0 02    .7        STA DELTA.TABLE,X
085D- E8                    INX
085E- D0 FA                 BNE .7      ...256 OF THEM
                  *---DELTA(KEY(I))=I-------------
0860- A0 00                 LDY #0           FOR I=0 TO KEYLEN
0862- B9 00 02    .8        LDA KBDBUF,Y   DELTA(K) = DISTANCE FROM LEFT END
0865- AA                    TAX                     OF RIGHT-MOST OCCURENCE OF
0866- 98                    TYA                     8-BIT VALUE "K" IN KEY.
0867- 9D D0 02              STA DELTA.TABLE,X
086A- C8                    INY              NEXT I
086B- C4 40                 CPY KEY.LENGTH
086D- 90 F3                 BCC .8
086F- F0 F1                 BEQ .8
                  *---ADJUST END OF SEARCH---------
0871- 38                    SEC
0872- A5 3E                 LDA A2L
0874- E5 40                 SBC KEY.LENGTH
0876- 85 3E                 STA A2L
0878- B0 02                 BCS SEARCH.LOOP
087A- C6 3F                 DEC A2L+1
                  *-------------------------------
                  SEARCH.LOOP
087C- A5 3D               LDA A1L+1     <<<DEBUG>>>
087E- A6 3C               LDX A1L       <<<DEBUG>>>
0880- 20 41 F9            JSR PRINTAX   <<<DEBUG>>>
0883- A9 AD               LDA #"-"      <<<DEBUG>>>
0885- 20 ED FD            JSR $FDED     <<<DEBUG>>>
0888- A5 3E                 LDA A2L          CHECK AGAINST
088A- C5 3C                 CMP A1L           UPPER BOUND
088C- A5 3F                 LDA A2L+1         FOR SEARCH
088E- E5 3D                 SBC A1L+1
0890- B0 01                 BCS .1           A1<=A2, NOT FINISHED
0892- 60                    RTS              A1>A2, FINISHED
                  *---COMPARE IN THIS POSITION-----
0893- A4 40       .1        LDY KEY.LENGTH   FOR I=KEYLEN TO 0
0895- B1 3C       .2        LDA (A1L),Y      CHECK BYTES FROM
0897- D9 00 02              CMP KBDBUF,Y        RIGHT TO LEFT
089A- D0 10                 BNE .3           ...DID NOT MATCH
089C- 88                    DEY              NEXT I
089D- 10 F6                 BPL .2
                  *---MATCH FOUND-----------------
089F- A6 3C                 LDX A1L          PRINT ADR
08A1- A5 3D                 LDA A1L+1        WHERE MATCH
08A3- 20 41 F9              JSR PRINTAX         WAS FOUND
08A6- 20 8E FD              JSR CROUT        NEW LINE
08A9- 4C B4 08              JMP .4
```

```
                *---ADVANCE SEARCH POINTER-------
08AC- AA        .3           TAX                    STRING CHAR JUST LOOKED AT
08AD- 98                     TYA                    I
08AE- 18                     CLC
08AF- FD D0 02               SBC DELTA.TABLE,X
08B2- 10 02                  BPL .5                 ...VALUE IS POSITIVE
08B4- A9 00     .4           LDA #0                 ...ADVANCE BY 1
08B6- 38        .5           SEC                    COMPENSATE
08B7- 65 3C                  ADC A1L
08B9- 85 3C                  STA A1L
08BB- 90 BF                  BCC SEARCH.LOOP
08BD- E6 3D                  INC A1L+1
08BF- D0 BB                  BNE SEARCH.LOOP        ...ALWAYS, UNLESS WE
08C1- 60                     RTS                    ...RAN OFF THE END OF MEMORY
                *-------------------------------
                NXTCHAR
08C2- B9 00 02               LDA KBDBUF,Y           NEXT ACTIVE CHAR
08C5- C8                     INY
08C6- 49 B0                  EOR #$B0               CONVERT ASCII TO DIGIT
08C8- C9 0A                  CMP #10                0..9?
08CA- 90 09                  BCC .1                 YES
08CC- 69 88                  ADC #$88               SHIFT RANGE FOR A-F TEST
08CE- C9 FA                  CMP #$FA               A..F?
08D0- B0 03                  BCS .1                 YES. EXIT CC
08D2- 38                     SEC                    NOT HEX CHAR
08D3- 88                     DEY                    BACK UP INDEX
08D4- 60                     RTS
08D5- 18        .1           CLC                    SIGNAL HEX CHAR
08D6- 60                     RTS
                *-------------------------------
08D7-           END     .BS $A00-*
                TEST.STRING
0A00- 58 58 58
0A03- 58 58 58
0A06- 58 43 4F
0A09- 43 41 43
0A0C- 41 43 41
0A0F- 43 41 43
0A12- 41 43 41
0A15- 43 41 43
0A18- 41 43 41
0A1B- 43 41 43
0A1E- 41 43                  .AS /XXXXXXXCOCACACACACACACACACACAC/
                * TRY A00.A1F^Y  434143414341 43
                * SHOULD GET A09-A0B-A0D-A0F-A11-A13-A15-A17-A19
                *-------------------------------
0A20- 41 20 53
0A23- 54 52 49
0A26- 4E 47 20
0A29- 53 45 41
0A2C- 52 43 48
0A2F- 49 4E 47
0A32- 20 45 58
0A35- 41 4D 50
0A38- 4C 45 20
0A3B- 43 4F 4E
0A3E- 53 49 53
0A41- 54 49 4E
0A44- 47 20 4F
0A47- 46 20 53
0A4A- 49 4D 50
0A4D- 4C 45 20
0A50- 54 45 58
0A53- 54        TS2     .AS /A STRING SEARCHING EXAMPLE CONSISTING OF SIMPLE TEXT/
                * TRY A20.A53^Y48494E47
                *-------------------------------
```

## Note About Alliance Computers

Back in January Alliance Computers advertised 65802's for $50.00, but couldn't fill the orders because the chips didn't exist yet. Some of their formerly unhappy customers tell me that their orders have now arrived, so Alliance is taking care of their customers. You can reach Alliance Computers at P.O. Box 408, Corona. NY 11368.

## Short Integer Square Root Subroutine.......Bob Sander-Cederlof

In some graphics situations you need a square root subroutine (it is probably the fault of Pythagoras). Since the screen coordinates are integers, a short and fast integer square root subroutine can be handy.

The following program is probably not in the "fast" category, but it is indeed short. It can produce the integer value of the square root of any integer from 0 through 65535. The program uses the method of subtracting successive odd numbers.

Every perfect square (N*N, where N is an integer) is the sum of a series of odd numbers from 1 through 2*N-1. Thus 4=1+3, 25=1+3+5+7+9, etc.

The program starts by subtracting 1, then 3, then 5, and so on until the remainder is negative. When the remainder goes negative, the last odd number subtracted was 2*N+1, so we can get the square root by dividing that odd number by 2.

I set up the routine so I could test it with an Applesoft program. You can POKE the low 8-bits of a number at 768 ($300), the high 8-bits at 769, and CALL 772. Upon return, PEEK(770)+256*PEEK(771) gives you the integer value of the square root.

I used a couple of tricks in the code. For one, the variable ODD is always an even number. Since I preface the subtraction with CLC, a "borrow" is assumed, so it has the effect of subtracting the odd number which is one larger than the even number in ODD. This save a LDA #1 instruction after line 1090.

In lines 1190-1230, I add 2 to the even number in ODD. But you can see that line 1200 is ADC #1. This adds 2 because carry happens to be set.

```
                  1000 *SAVE S.SQRT16
                  1010 *--------------------------------
                  1020        .OR $300
0300-             1030 ARG    .BS 2
0302-             1040 ODD    .BS 2
                  1050 *--------------------------------
0304- AE 01 03    1060 SQRT   LDX ARG+1      X = HI BYTE   HI
0307- AC 00 03    1070        LDY ARG        Y = LO BYTE   LO
030A- A9 00       1080        LDA #0         START ODD=0
030C- 8D 03 03    1090        STA ODD+1
030F- 8D 02 03    1100 .1     STA ODD
0312- 18          1110        CLC            BORROW ON. SUBTRACT (ODD+1)
0313- 98          1120        TYA            LO
0314- ED 02 03    1130        SBC ODD
0317- A8          1140        TAY
0318- 8A          1150        TXA            HI
0319- ED 03 03    1160        SBC ODD+1
031C- AA          1170        TAX
031D- 90 0C       1180        BCC .2         ...ODD>REMAINDER. FINISHED
031F- AD 02 03    1190        LDA ODD        CARRY SET, ADD 2 TO ODD
0322- 69 01       1200        ADC #1
0324- 90 E9       1210        BCC .1         ...NEXT
0326- EE 03 03    1220        INC ODD+1
0329- D0 E4       1230        BNE .1         ...ALWAYS
032B- 4E 03 03    1240 .2     LSR ODD+1      SQRT IS (ODD/2)
032E- 6E 02 03    1250        ROR ODD
0331- 60          1260        RTS
                  1270 *--------------------------------
```

Note on the TXS instruction in the 65802...Bob Sander-Cederlof

Sandy Greenfarb wrote the other day that he had received a 65802 and plugged it into his Basis 108 with success.

He has been trying various permutations of the new opcodes and modes, and discovered some stones are better left unturned:

> "The following programs should both print the letter "A" on the screen. However, the one on the left works, while the one on the right hangs up the computer."

```
Works            Hangs Up
--------         --------
CLC              CLC
XCE              XCE
LDA #"A          LDA #"A
JSR $FDF0        JSR $FDED
SEC              SEC
XCE              XCE
RTS              RTS
```

The only difference in the two programs is that the unsuccessful one weaves its way through DOS. I looked at the DOS code it goes through, and at first glance it appears there should be NO PROBLEMS associated with executing all this code in 65802 mode, since both 16-bit modes are off.

However, for some reason it still hangs up. Actually, it might not always hang: it depends on what is in page zero at the corresponding position as the stack pointer in page one.

I do not know why, but the TXS instruction transfers the entire 16-bit value of X to S when you are in the 65802 mode, regardless of the status of the M and X bits. Since M and X are both 1, the high byte of the X-register is 00. Therefore the TXS instruction at $9FB9 in DOS clears the high byte of the S-register. The RTS at $9FC4 then uses a return address from page zero, rather than page one.

I tried various experiments to see how TXS and TSX worked, and also examined TXA and TAX. In my humble opinion, the 65802 is inconsistent here. If you are in 65802 mode with M and X = 1, TXA does not modify the high byte of the A-register. This is what I expect and what I want. But TXS does modify the high byte of the S-register, contrary to my expectations.

Of course, as long as you know exactly how the chip works it really doesn't matter a lot. The problems come when we ASSUME we know how it works, but are wrong. The best antidote for these kind of assumptions, at least until a definitive reference manual for the chip is published, is trial and error.

I have had my 65802 for about six months now, and still have had no problems whatsoever with compatibility as long as I stay in normal 6502 mode. If I leave it in 65802 and go charging through a program written for the 6502 mode, I expect I will run into trouble.

---

Interrupt Trace..............................Charles H. Putney
Dublin, Ireland

Have you ever wondered what's happening when the Apple goes off into nothingness? If your answer is yes, then this short utility will help you find out.

I was recently debugging an assembly language program and ran into this problem The program seemed to work for almost all the input data, but occasionally would hang. After several frustrating hours trying to simulate the event, I decided that an interrupt trace utility would solve my problems. Later when I had this utility working, it was easy to see why the program was hanging.

This utility consists of a pushbutton addition to the Apple which connects to the interrupt request line (IRQ) of the 6502 and an interrupt service routine which is in page three. When the interrupt pushbutton is depressed, the interrupt service routine displays the program counter and all the registers on the bottom line of the screen. It also displays a flashing cursor and waits for an "S" or "G" from the keyboard to stop or resume execution.

I have mounted a pushbutton switch at the upper right hand side of the keyboard in the center of the styling surface. For a temporary installation I suggest leaving the pushbutton on a flexible lead. The wiring is easily done with 30 gage wire wrap wire. One side of the pushbutton is connected to ground. You may solder a wire to any convenient ground point on the top of the circuit board. Or, for a temporary installation, you could stick a wire into pin 8 of the game I/O connector.

The other side of the pushbutton is connected to the IRQ signal. I found that signal at pin 4 of the 6502. Remove the 6502 from the socket and strip the insulation from the end of the 30 gage wire. Insert it in the socket for the 6502 in pin 4 and replace the 6502 to retain the wire. Route the wire along the chips for a neat installation.

For a temporary hookup, Bob S-C suggests folding a 3-by-5 card in half, and triming it so that the folded edge just fits into an empty slot. Then, while power to the Apple is off, slip one wire into the space between the card and pin 26 (ground) and the other wire between the card and pin 30 (IRQ). Both of these wires will be on the power-supply side of the card: pin 26 is at the back edge, and pin 30 is the fifth from the back. Once the wires are inserted, you may wish to tape them down.

Enter the routine at address $300 and BSAVE it. When you want to debug a hanging program, first BRUN the INTERRUPT TRACE utility. This installs the utility at address $300 to $3CA. Pressing the pushbutton will cause an immediate display of the current program counter and registers. The utility will wait with a blinking cursor for a "G" or "S" from the keyboard to continue or enter monitor.

Sometimes the program you're investigating may not respond to

the pushbutton. This is because somewhere in the program interrupts have been disabled with the SEI command ($78). You must search through the entire program and replace these with a CLI instruction ($58). Make sure that each $78 found is not data in the program and is a valid instruction before you replace it.

The next time that you have a problem with your Apple "hanging" for no apparent reason, use this utility to see where the 6502 "is". It may help solve those "hard to debug programs".

When you run the program, the SETUP routine (from $300 to $30B) sets the interrupt vector location and then enables interrupts. When the pushbutton is depressed, the IRQ line (pin 4 on the 6502) is pulled low. At the completion of the current instruction, the program counter high, program counter low, and processor status are pushed on the stack. Interrupt disable is automatically set and the program counter is loaded with the contents of $FFFE and $FFFF. In the Autostart monitor ROM the program counter is set to $FA40 where the monitor interrupt service routine is located. (In the old monitor the identical routine is at $FA86) This routine saves the accumulator in $45 and examines the processor status register to see if the interrupt was caused by a BRK command. Remember, the BRK command shares the same vector location with the interrupt for software simulation of interrupts. If the interrupt was not caused by BREAK then a JMP indirect to location ($3FE) is performed.

Lines 1280-1290 save the X- and Y-registers. The accumulator has already been saved by the monitor interrupt routine.

Lines 1300-1350 copy the register display titles to the bottom of the screen. Of course, if your program happened to be in one of the full-screen graphics modes, this line will not be visible. If you have a //e, you can add code to sense the graphics mode, save it, switch to text mode; then you will have to restore it all when you type "G" to continue after the interrupt. The new enhanced //e ROMs automatically handle saving and restoring all the bank switched memory, but they still leave the graphics modes up to the programmer.

Lines 1360-1510 convert the values of the five registers and store them into the bottom line. I add 3 to the S-register value before displaying it, so you see the value before the IRQ code pushed PC and S onto the stack. I start with the Y-register pointing at the point on the bottom line where the A-register should be displayed. The DISPLAY.HEX subroutine advances the Y-register by 5, so it is always ready for displaying the next register

Lines 1520-1590 display the PC-register. This value is taken from the stack, where the IRQ automatically saved it.

Lines 1600-1750 wait for you to type "G" or "S". While waiting, the last character on the bottom line is flashed to remind you to type. If you type "G", lines 1760-1800 restore the registers and return to the interrupted program. If you type "S", line 1820 takes you to the monitor.

```
                   1000 *SAVE S.IRQ TRAPPER
                   1010 *--------------------------------
                   1020 *       INTERRUPT TRACE UTILITY
                   1030 *
                   1040 *       BY: CHARLES H  PUTNEY
                   1050 *           18 QUINNS ROAD
                   1060 *           SHANKILL
                   1070 *           COUNTY DUBLIN
                   1080 *           IRELAND
                   1090 *--------------------------------
45-                1100 A.REG       .EQ $45       A-REG SAVE AREA USED BY MONITOR
0100-              1110 STACK       .EQ $100      STACK PAGE
03FE-              1120 INTVEC      .EQ $3FE      INTERRUPT VECTOR
07D0-              1130 BOTTOM.LINE .EQ $7D0      LINE 24 OF TEXT SCREEN
                   1140 *--------------------------------
C000-              1150 KEYBD  .EQ $C000    KEYBOARD DATA
C010-              1160 KEYSTB .EQ $C010    KEYBOARD STROBE
FF69-              1170 MNTR   .EQ $FF69    MONITOR ENTRY POINT (CALL -151)
                   1180 *--------------------------------
                   1190        .OR $300     PAGE THREE
                   1200 *--------------------------------
0300- A9 0C        1210 SETUP  LDA #INT     LOAD IRQ VECTOR
0302- 8D FE 03     1220        STA INTVEC   LOW BYTE
0305- A9 03        1230        LDA /INT
0307- 8D FF 03     1240        STA INTVEC+1 HIGH BYTE
030A- 58           1250        CLI          ALLOW IRQ'S
030B- 60           1260        RTS
                   1270 *--------------------------------
030C- 8E C5 03     1280 INT    STX XREG     SAVE X (A-REG SAVED BY MONITOR)
030F- 8C C6 03     1290        STY YREG     SAVE Y
                   1300 *---DISPLAY REG TITLES-----------
0312- A2 27        1310        LDX #39      PUT UP MESSAGE LINE
0314- BD 9D 03     1320 .1     LDA TITLES.X  GET MESSAGE CHAR
0317- 9D D0 07     1330        STA BOTTOM.LINE,X   PUT ON SCREEN
031A- CA           1340        DEX
031B- 10 F7        1350        BPL .1       DONE ?
                   1360 *---DISPLAY REG VALUES-----------
031D- A0 0A        1370        LDY #10      START OF REG DISPLAY AREA
031F- A5 45        1380        LDA A.REG    ...A-REG
0321- 20 7B 03     1390        JSR DISPLAY.HEX
0324- AD C5 03     1400        LDA XREG     ...X-REG
0327- 20 7B 03     1410        JSR DISPLAY.HEX
032A- AD C6 03     1420        LDA YREG     ...Y-REG
032D- 20 7B 03     1430        JSR DISPLAY.HEX
0330- BA           1440        TSX          GET STACK POINTER
0331- E8           1450        INX          POINT AT PROCESSOR STATUS
0332- BD 00 01     1460        LDA STACK,X  ...P-REG
0335- 20 7B 03     1470        JSR DISPLAY.HEX
0338- E8           1480        INX          ADJUST S-REG
0339- E8           1490        INX
033A- 8A           1500        TXA          ...S-REG AS WAS BEFORE INTERRUPT
033B- 20 7B 03     1510        JSR DISPLAY.HEX
                   1520 *---DISPLAY PC-REG---------------
033E- A0 00        1530        LDY #0       START OF PC-REG DISPLAY
0340- BD 00 01     1540        LDA STACK,X  GET PC HIBYTE
0343- 20 7B 03     1550        JSR DISPLAY.HEX
0346- CA           1560        DEX
0347- A0 02        1570        LDY #2
0349- BD 00 01     1580        LDA STACK,X  GET PC LOBYTE
034C- 20 7B 03     1590        JSR DISPLAY.HEX
                   1600 *---WAIT FOR "S" OR "G"----------
034F- AD 00 C0     1610 .2     LDA KEYBD    KEY PRESSED ?
0352- 10 0B        1620        BPL .3       NO
0354- 8D 10 C0     1630        STA KEYSTB   CLEAR THE KEY
0357- C9 C7        1640        CMP #"G"     GO AHEAD ?
0359- F0 14        1650        BEQ .4       YES
035B- C9 D3        1660        CMP #"S"     STOP ?
035D- F0 19        1670        BEQ .5       YES
035F- CA           1680 .3     DEX          BLINK CURSOR
0360- D0 FD        1690        BNE .3
0362- 88           1700        DEY          LONGER DELAY
0363- D0 FA        1710        BNE .3
0365- AD F7 07     1720        LDA BOTTOM.LINE+39  LAST CHAR ON SCREEN
0368- 49 80        1730        EOR #$80     INVERT IT
036A- 8D F7 07     1740        STA BOTTOM.LINE+39  REPLACE IT
036D- D0 E0        1750        BNE .2       BRANCH ALWAYS
```

```
                  *---"G" TYPED, RETURN-----------
036F- A5 45       .4      LDA A.REG      RESTORE THE REGISTERS
0371- AE C5 03            LDX XREG
0374- AC C6 03            LDY YREG
0377- 40                  RTI            BACK TO WORK
                  *---"S" TYPED, SO STOP----------
0378- 4C 69 FF    .5      JMP MNTR       ENTER THE MONITOR
                  *-------------------------------
                  DISPLAY.HEX
037B- 48                  PHA            SAVE THE A-REG
037C- 4A                  LSR            SHIFT INTO LOWER NIBBLE
037D- 4A                  LSR
037E- 4A                  LSR
037F- 4A                  LSR
0380- 20 93 03            JSR DIGIT      MAKE IT A DIGIT
0383- 99 D0 07            STA BOTTOM.LINE,Y   SHOW HIGH NIBBLE
0386- 68                  PLA            GET A-REG AGAIN
0387- 29 0F               AND #$0F       MASK IT
0389- 20 93 03            JSR DIGIT      MAKE IT A DIGIT
038C- 99 D0 07            STA BOTTOM.LINE,Y   SHOW LOWER NIBBLE
038F- C8                  INY
0390- C8                  INY
0391- C8                  INY
0392- 60                  RTS
                  *-------------------------------
0393- C8          DIGIT   INY
0394- 09 B0               ORA #$B0       ADD NUMBER ZERO
0396- C9 BA               CMP #$BA       IS IT A LETTER
0398- 90 02               BCC .1         NO - DONE
039A- 69 06               ADC #$6        6 PLUS CARRY MAKES A
039C- 60          .1      RTS
                  *-------------------------------
039D- A0 A0 A0
03A0- A0 A0 AD
03A3- A0 A0 A0
03A6- C1 BD A0
03A9- A0 A0 D8
03AC- BD A0 A0
03AF- A0 D9 BD
03B2- A0 A0 A0
03B5- D0 BD A0
03B8- A0 A0 D3
03BB- BD A0 A0
03BE- A0 A0 A0
03C1- A0 A0 A0
03C4- A0          TITLES .AS -/     -    A=   X=   Y=   P=   S=        /
                  *-------------------------------
03C5- 00          XREG    .DA #*-*       X REGISTER SAVE AREA
03C6- 00          YREG    .DA #*-*       Y REGISTER SAVE AREA
```

## Improving the Single-Byte Converter...............Bruce Love
New Zealand

Bob's single byte converter (see Jan 85 issue, pages 31-32) can be shortened by one byte. The left column is from Bob's code. the right a shorter version:

```
.1 LDX #"0"                    .1 LDX #"0"-1
.2 CMP DECTBL,Y                   SEC
   BCC .3                      .2 SBC DECTBL,Y
   SBC DECTBL,Y                   INX
   INX                            BCS .2
   BNE .2                         ADC DECTBL,Y
```

I also tried a different approach, using the decimal mode to count tens, then printing the tens as a hex value with the monitor routine at $FDDA and the remainder (units digit) with $FDED. This routine takes longer time, but does not need to use the X-register.

```
00-                BYTE   .EQ $00
FDED-              COUT   .EQ $FDED
FD8E-              CROUT  .EQ $FD8E
FDDA-              PRBYTE .EQ $FDDA
                   *-------------------------------
0800- A9 00        P      LDA #0
0802- 85 00               STA BYTE
0804- 20 11 08     .1     JSR PRINT.000.255
0807- 20 8E FD            JSR CROUT
080A- E6 00               INC BYTE
080C- A5 00               LDA BYTE
080E- D0 F4               BNE .1
0810- 60                  RTS
                   *-------------------------------
                   PRINT.000.255
0811- A0 00               LDY #0
0813- 38                  SEC
0814- E9 0A        .1     SBC #10
0816- 08                  PHP
0817- 48                  PHA
0818- 98                  TYA
0819- F8                  SED
081A- 69 00               ADC #0
081C- A8                  TAY
081D- 68                  PLA
081E- 28                  PLP
081F- B0 F3               BCS .1
0821- 69 BA               ADC #"0+10
0823- 48                  PHA
0824- 98                  TYA
0825- 20 DA FD            JSR PRBYTE
0828- 68                  PLA
0829- 4C ED FD            JMP COUT
```

## AppleVisions, a Glimpse

Here is a very elementary introduction to Apple Assembly Language programming by that old master Bob Bishop, along with Linda Grossberger and Harry Vertelney. This 150+ page book and its companion diskette gently and humorously guide the beginning programmer into the realm of machine code. A "Cardboard Computer" introduces the concepts of registers, machine instructions, addressing, and branching. This background is then applied to the Apple's 6502 and the surrounding computer. AppleVisions is a nice place for the absolute beginner to start, especially the younger programmers interested in finding out what assembly language is.

AppleVisions. Addison-Wesley, 1985. $39.95 including diskette.

Two ROM Sets in One Apple //e.............Bob Sander-Cederlof

If and when you decide to upgrade to the new enhanced //e ROMs (which Apple sells for $70 along with a 65C02), you will probably have to turn your old ROMs over to the store that makes the switch. Reportedly, Apple is binding the stores with a contract that forces them to collect all the old chips.

That is VERY unfortunate. It could lead to wild shouting and panic, when you discover some of your favorite old software no longer works.

The upgrade consists of three parts:

* the new processor chip (65C02), which is nice but not especially useful until software which uses its new features becomes available;

* a new character generator ROM which includes special characters for icons and line drawing in text mode (called the "mouse" characters).

* new CD and EF ROMs which upgrade the firmware.

The new firmware does NOT use any of the new features in the 65C02, so you could use it without the new cpu chip. Furthermore, there is no absolute requirement to have the new character generator installed. The new firmware is much better than the old, having lost some bugs and speeded up the 80-column scrolling and added lower-case support to Applesoft (among other things). It is compatible with the 6502, the 65C02, and the new 65802.

I personally do not yet have any use for the mouse characters, and do not expect to. Don Lancaster, in the June 1985 issue of "Computer Shopper", tells how to connect a 2764 EPROM in the character generator socket. The 2764 can hold two complete character sets, because it has twice the capacity of the 2732 normally in that socket. However, the socket has only 24 holes and the 2764 has 28 pins! Don shows how to wire this up with a socket adapter. and use a toggle switch to select either half.

And now Apple has "sort of" released an even more enhanced set of firmware, with debugging stuff built in. You may not see them on the open market for some time, but I like them even better than the standard enhanced ROMs. The "debug" ROMs add an absolute RESET (ctrl-RESET with solid apple), 16-byte hex display in the monitor when in 80-column mode, display of both hex and ASCII values of each byte in a memory dump, and the ability to use all monitor commands on both main and auxiliary memory. The disassembler and miniassembler are both present, and enhanced to include the 65C02 extensions.

The CD and EF ROM sockets are compatible with 2764 EPROMs. You can also use 27128s, which have twice the space. Pin 26 on the 2764 is always tied to +5 volts. On a 27128, pin 26 selects the top or bottom half of the 16K bytes inside. You can burn one set of firmware in one half, and the other set in the other

half. Then bend out pin 26 a little, so that it does not go into the socket when you insert the chip. Attach a clip lead to the bent-out pin, and connect the other end to either +5 volts or ground, to select the half you want at any given time.

You can connect it to a toggle switch, or just stick the bare end of a wire into the game paddle connector. If you use the game socket on the motherboard, pin 1 is +5 volts and pin 8 is ground. Or stick a wire into one of the annunciator outputs (pins 12, 13, 14, and 15) so you can flip back and forth between firmware sets by software control.

It can be a little tricky to make a copy of the ROM firmware and get it into RAM or on a disk, so that you can later burn it in your own EPROM. Especially in the Cxxx part. My approach, since I have more than one Apple, is to put my SCRG PromGramer card in a different machine. Then one by one I can read the //e ROMs and burn them into the appropriate 27128s. This a lot faster than trying to figure out how to flip all the //e soft switches so as to get at the different banks of Cx ROM code.

I have recently seen 27128s priced as low as $5 and as high as $20, in the back of Byte magazine. It is well worth it to invest in a PromGramer, at $140, and an EPROM eraser ($50 to $100 from Logical Devices in Florida, see Byte ads). You can keep your Apple standard for commercial software, and still have your own private firmware on the motherboard at the flip of a switch!

Just after finishing this article I happened to be looking up something else in Understanding the Apple //e, by Jim Sather. There, big as life, on page 6-11, is a description of the same method for installing modified ROMs.

## A CALL Utility for Applesoft..................David C. Johnson

Applied Engineering

Anyone who has ever used Applesoft eventually realizes that the most powerful statement in the language is CALL. It allows you to get to the Monitor for instance (however, Extended Debugging Monitor users have a better way). When writing a program in BASIC, you invariably will want to do something that is at best difficult and often impossible to code using the other Applesoft statements.

The solution to this type of situation is to speak to your Apple in its native tongue. There are several way this can be done. Ampersand (&) routines are a popular technique. The USR( function even has its uses. The most logical way, for me, is the CALL statement.

Using CALL neatly transfers control from the Applesoft interpreter to whatever you want to do in machine language. The one disadvantage to CALL is that the processor's registers do not contain useful data when your machine code gets control.

The CALL utility presented in this article will allow you to specify, as part of the CALL statement, the contents of any or all of the registers upon entry of your machine language subroutine. You assign the register contents with LET-like structures. Obviously you can only fit an 8 bit value into the 8 bit registers and the program counter value will probably be a 16 bit number. Here's how the CALL statement should be written:

```
CALL 768,PC=word,A=byte,X=byte,Y=byte,P=byte
```

The expressions "word" and "byte" may be any valid Applesoft numeric expression. This is because the utility calls routines internal to Applesoft to evaluate the expressions. If an expression results in a value larger than the register to which it is being assigned, or isn't numeric, or is invalid, you will get one of the usual errors. The commas shown separating the register assignments are required (syntax error if comma missing). The equals characters ("=") are also required. The register names ( PC, A, X, Y, & P) must be upper case on older Apples, while the newer firmware will convert lower case for you (or in spite of you). The register assignments may appear, after the first comma, in any order and need not all be specified. Unspecified registers will be loaded with their last used value. Previously unused registers default to zero, except the P-register which defaults to $04 in order to set the interrupt disable flag.

The program is well commented, but I'll add one more note of caution. Readers with Apples containing reqular 6502s (not 65C02s or 65802s) should avoid re-assembling the code with the label PC.Sav's bytes falling across a page boundary ($XXFF).

The program was written using the ProDOS version of the S-C Macro Assembler 2.0, while I was beta testing it for Bob. It works GREAT!

```
                *SAVE S.CALL.UTIL

                * 6/13/85 dcj

                * CALL 768{,pc=word,a=byte,x=byte,y=byte,p=byte}

                *-------------------------------

                        .OR $300
                        .TF CU

D0-             EQ.TOK  .EQ $D0         Applesoft '=' token

B1-             CHRGET  .EQ $B1 -$C8    advance TXTPTR & fetch chr
B7-             CHRGOT  .EQ $B7         just fetch chr

DD67-           FRMNUM  .EQ $DD67       evaluate FP expression (FAC)
DEC0-           SYNCHR  .EQ $DEC0       require chr in Acc syntax @ TXTPTR
DEC9-           SYNERR  .EQ $DEC9       syntax error
E6F8-           GETBYT  .EQ $E6F8       evaluate 8 bits @ TXTPTR (X-reg)
E752-           GETADR  .EQ $E752       convert FAC to 16 bits in Acc & Y-reg

                *-------------------------------

                CALL.UTIL

0300- 20 B7 00          JSR CHRGOT      get chr after call adr expression
0303- C9 2C             CMP #','        comma indicates more stuff follows
0305- F0 11             BEQ .1          =>go continue parsing
0307- AD 7D 03          LDA P.SAV       load registers
030A- 48                PHA              (P-reg via stack)
030B- AD 7A 03          LDA ACC.SAV
030E- AE 7B 03          LDX X.SAV
0311- AC 7C 03          LDY Y.SAV
0314- 28                PLP
0315- 6C 7E 03          JMP (PC.SAV) go 4 it!

                * we got something to parse

0318- 20 B1 00  .1      JSR CHRGET      get chr after comma
031B- C9 41             CMP #'A'        (as in 'Acc')
031D- F0 0F             BEQ .2          =>go get '=' & byte for Acc
031F- C9 58             CMP #'X'        (as in 'X-reg')
0321- F0 13             BEQ .3          =>go get '=' & byte for X-reg
0323- C9 59             CMP #'Y'        (as in 'Y-reg')
0325- F0 17             BEQ .4          =>go get '=' & byte for Y-reg
0327- C9 50             CMP #'P'        (as in P-reg or Program Counter)
0329- F0 1B             BEQ .5          =>go get '=' or 'C='...
032B- 4C C9 DE          JMP SYNERR      razz

                * pickup Acc byte

032E- 20 6A 03  .2      JSR .7          require '=' (@ next) & fetch byte exp
0331- 8E 7A 03          STX ACC.SAV     stuff it
0334- 50 CA             BVC CALL.UTIL ...always

                * pickup X-reg byte

0336- 20 6A 03  .3      JSR .7          require '=' (@ next) & fetch byte exp
0339- 8E 7B 03          STX X.SAV       stuff it
033C- 50 C2             BVC CALL.UTIL ...always

                * pickup Y-reg byte

033E- 20 6A 03  .4      JSR .7          require '=' (@ next) & fetch byte exp
0341- 8E 7C 03          STX Y.SAV       stuff it
0344- 50 BA             BVC CALL.UTIL ...always

                * Finish parsing 'P=' or 'PC='

0346- 20 B1 00  .5      JSR CHRGET      advance to next chr position & fetch it
0349- C9 43             CMP #'C'        (as in 'Program Counter')
034B- F0 0B             BEQ .6          =>go get '=' & 16 bits for PC

                * pickup P-reg byte

034D- 20 75 03          JSR .10         require '=' @ current chr position
0350- 20 6D 03          JSR .8          fetch byte expression
0353- 8E 7D 03          STX P.SAV       stuff it
0356- 50 A8             BVC CALL.UTIL ...always
```

```
                     1800
                     1810 * pickup PC word
                     1820
0358- 20 72 03  1830 .6      JSR .9        require '=' @ next chr position
035B- 20 67 DD  1840         JSR FRMNUM    fletch FP expression
035E- 20 52 E7  1850         JSR GETADR    convert FP expression to Acc & Y-reg
0361- 8C 7E 03  1860         STY PC.SAV    stuff 'em
0364- 8D 7F 03  1870         STA PC.SAV+1
0367- 4C 00 03  1880         JMP CALL.UTIL no flag known...
                     1890
036A- 20 72 03  1900 .7      JSR .9        require '=' @ next chr position
                     1910
036D- 20 F8 E6  1920 .8      JSR GETBYT    fetch byte expression (2 X-reg)
0370- B8        1930         CLV           to force branch
0371- 60        1940         RTS
                     1950
0372- 20 B1 00  1960 .9      JSR CHRGET    1st advance to next chr position
                     1970
0375- A9 D0     1980 .10     LDA #EQ.TOK   require '=' before register expressions
0377- 4C C0 DE  1990         JMP SYNCHR     (SYNTAX ERROR IF '=' NOT FOUND)
                     2000
                     2010 *--------------------------------
                     2020
037A- 00        2030 ACC.SAV    .DA #$00
037B- 00        2040 X.SAV      .DA #$00
037C- 00        2050 Y.SAV      .DA #$00
037D- 04        2060 P.SAV      .DA #$04
037E- 00 00     2070 PC.SAV     .DA $0000
                     2080
                     2090 *--------------------------------
```

## Some Final DP18 Subroutines...............Bob Sander-Cederlof

Gerald Ferrier (Princeton, Minnesota) wrote to point out that we somewhow omitted a double-handful of subroutines from our lengthy series on 18-digit arithmetic for Applesoft. With apologies to you all, and thanks to Gerald, here they are:

```
1000 *SAVE DP18.MOVE.SUBS
2020 *--------------------------------
2030 *      MOVE (Y,A) INTO DAC. YA IS UNPACKED
2040 *--------------------------------
2050 MOVE.YA.DAC.1
2060        STA PNTR
2070        STY PNTR+1
2080        LDY #10        MOVE 11 BYTES
2090 .1     LDA (PNTR),Y
2100        STA DAC,Y
2110        DEY
2120        BPL .1
2130        LDA DAC.EXPONENT
2140        STA DAC.SIGN
2150        AND #$7F
2160        STA DAC.EXPONENT
2170        RTS
2180 *--------------------------------
2190 *      MOVE (Y,A) INTO ARG. YA IS UNPACKED
2200 *--------------------------------
2210 MOVE.YA.ARG.1
2220        STA PNTR
2230        STY PNTR+1
2240        LDY #10        MOVE 11 BYTES
2250 .1     LDA (PNTR),Y
2260        STA ARG,Y
2270        DEY
2280        BPL .1
2290        LDA ARG.EXPONENT
2300        STA ARG.SIGN
2310        AND #$7F
2320        STA ARG.EXPONENT
2330        RTS
2340 *--------------------------------
2350 *      MOVE DAC TO (Y,A) WITHOUT PACKING
2360 *--------------------------------
2370 MOVE.DAC.YA.1
2380        STA PNTR
2390        STY PNTR+1
2400        LDA DAC.EXPONENT
2410        BPL .0
2420        JMP DAC.YA.O.U  OVER- OR UNDER-FLOW
2430 .0     BIT DAC.SIGN
2440        BPL .1         POSITIVE
2450        ORA #$80       NEGATIVE
2460 .1     LDY #0
2470 .2     STA (PNTR),Y
2480        INY
2490        LDA DAC,Y
2500        CPY #11        11 BYTES
2510        BCC .2
2520        RTS
2530 *-----------------------
2540 MOVE.DAC.TEMP1
2550        LDA #DP.TEMP1
2560        LDY /DP.TEMP1
2570        JMP MOVE.DAC.YA.1
2580 *-----------------------
2590 MOVE.TEMP1.ARG
2600        LDA #DP.TEMP1
2610        LDY /DP.TEMP1
2620        JMP MOVE.YA.ARG.1
2630 *-----------------------
2640 MOVE.TEMP1.DAC
2650        LDA #DP.TEMP1
2660        LDY /DP.TEMP1
2670        JMP MOVE.YA.DAC.1
2680 *-----------------------
2690 MOVE.DAC.TEMP2
2700        LDA #DP.TEMP2
2710        LDY /DP.TEMP2
2720        JMP MOVE.DAC.YA.1
2730 *-----------------------
2740 MOVE.TEMP2.DAC
2750        LDA #DP.TEMP2
2760        LDY /DP.TEMP2
2770        JMP MOVE.YA.DAC.1
2780 *-----------------------
2790 MOVE.TEMP2.ARG
2800        LDA #DP.TEMP2
2810        LDY /DP.TEMP2
2820        JMP MOVE.YA.ARG.1
2830 *-----------------------
2840 MOVE.TEMP3.DAC
2850        LDA #DP.TEMP3
2860        LDY /DP.TEMP3
2870        JMP MOVE.YA.DAC.1
2880 *-----------------------
2890 MOVE.TEMP3.ARG
2900        LDA #DP.TEMP3
2910        LDY /DP.TEMP3
2920        JMP MOVE.YA.ARG.1
2930 *-----------------------
2940 MOVE.DAC.TEMP3
2950        LDA #DP.TEMP3
2960        LDY /DP.TEMP3
2970        JMP MOVE.DAC.YA.1
2980 *-----------------------
```

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $14 postage for other countries. Back issues are available for $1.80 each (other countries add $1 per back issue for postage).